ACOTRIS : Real Time and Model Checking

Michel NAKHLE, CS
Thierry GAUTIER, IRISA/INRIA
Charles MODIGUY, CS

- **Web :**

http://www.acotris.c-s.fr/

First part : Contour of the Project

- **RNTL** Project
- Objectives
- Needs
- General Architecture

1st part – **ACOTRIS Project**

- **RNTL** Project (AAP2000/CFP2000) ; supported by **M.R.** (Ministry of Research)
- **Partners** : CS, CEA-List, MBDA France, INRIA-IRISA, SITIA
- Propose a **Methodology** and a **Systemic Approach** (+ tools support) :
  - Integrating methods for **formal Checking/Validation** & **co-design**
  - Independent of any (life) cycle
  - Adapted to the approaches used by the majority of the industrialists
  - Taking into account, by "simple" means,
    - o "Functional or Structural" & "Logic and temporal" **needs**,
    - o **Architecture** constraints (hardware),
  - Which Allows to rationalize the development phases of **R**eal **T**ime **E**mbedded **S**ystems

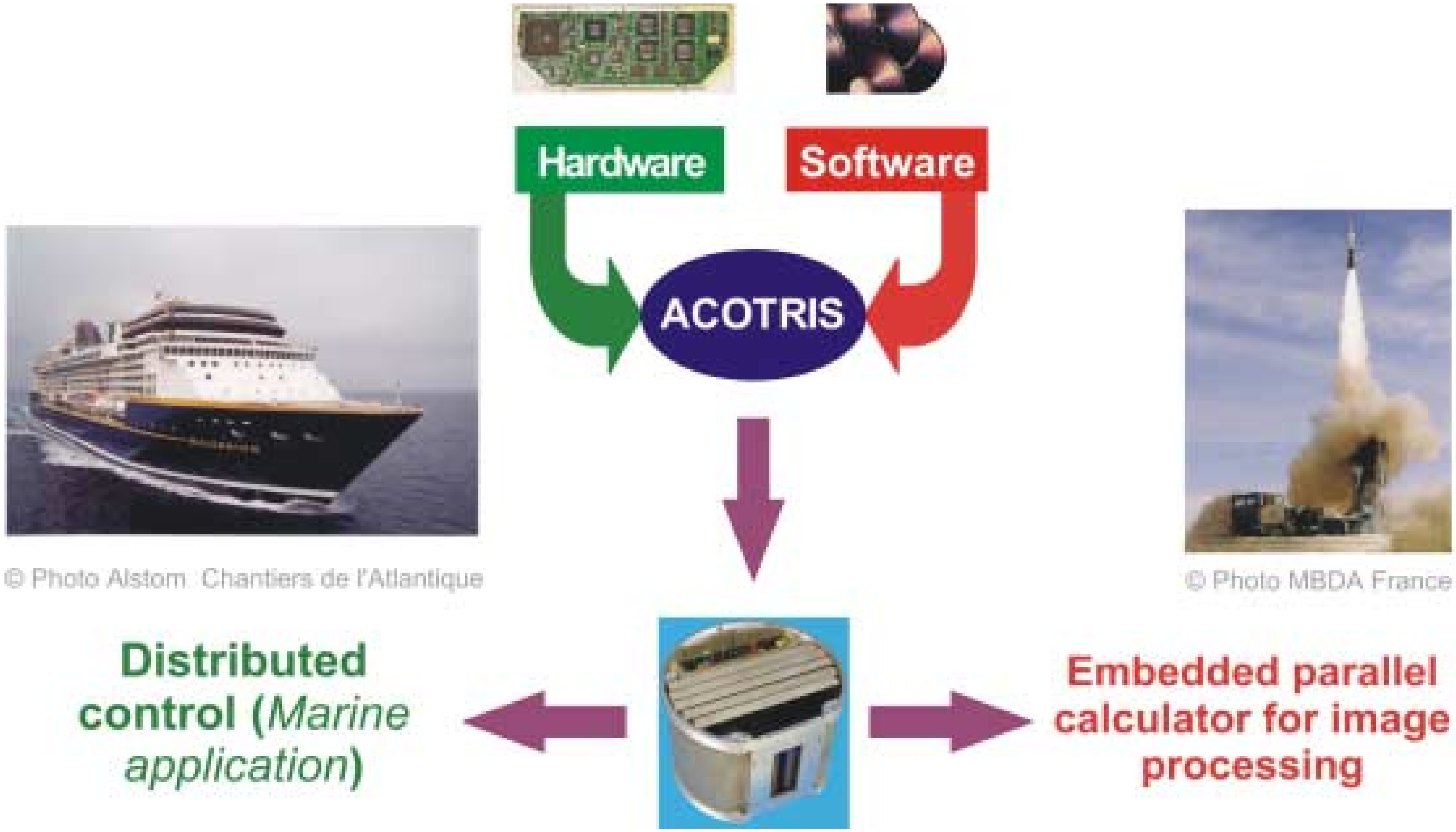
1st part – **Objectives**

- *To* Help with the complete specification of the need, and the Design of Real Time applications while integrating :
  - An Analysis and Design Methodology based on a standard asynchronous formalism **(*UML* *with* *ACCORD method* ↔ *CEA*)**
  - A Design and Realization Methodology based on the synchronous model **(*SIGNAL* *and* *AAA/SynDEx method* ↔ *INRIA*)**

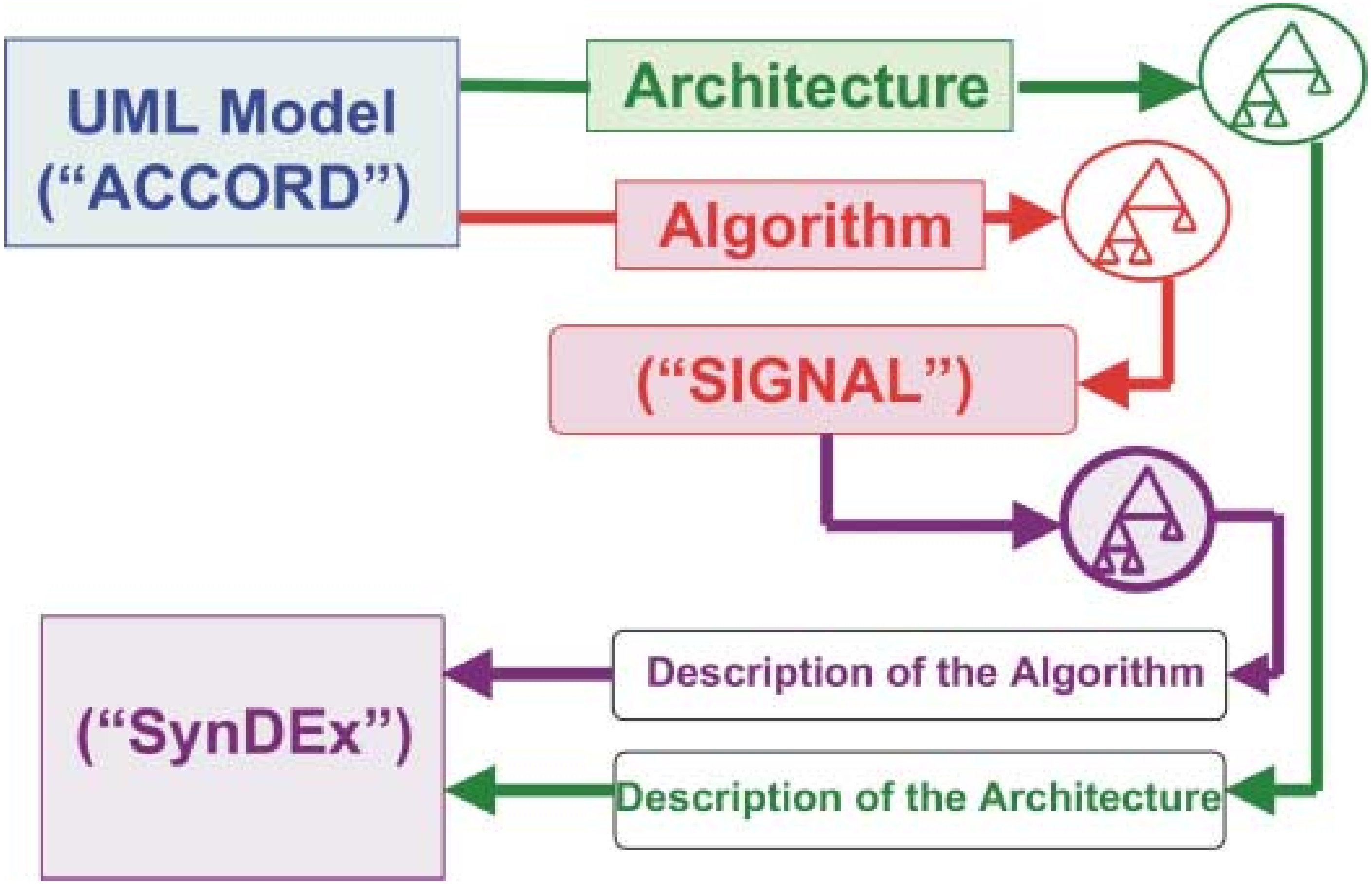
- *In order* To assist the designers of Real Time multi-task applications with strong parallelism during the co-design process by a quasi complete automation of this process
  - **Adapt and connect the existing tools (develop "footbridges")**

1st part - Expression of the need

- Evaluate and validate the technical solutions on two applications :
  - Embedded parallel calculator for image processing
  - Distributed control within a marine application



1st part – **General Architecture**



1st part – **Anticipated profits**

- **Advantages offered by UML :**
  - Modularity of the specifications
  - Problems separation
  - Components re-usability (heritage…)
  - Support to the refinement and the legibility of specifications
  - Coherent description of various aspects of the system of which :
    - Data Representation
    - Concurrence
    - Expression of the responsibilities in the system

- **Advantages offered by the synchronous languages :**
  - Parallelism of the languages (expression) and "centralized code" generation (Compilers)
  - Help to program development :
    - Numerical values
    - Input/output events
    - Optimisation
  - Conformity of the implementation to the specification on the Models level ⇔ SAFETY

Second part

1) Boolean Abstraction

- SIGNAL Language
- Boolean Abstraction of a SIGNAL program
- Polynomial Dynamical Systems

2) Verification of properties

- Liveness
- Invariance and invariance under control
- Reachability
- Attractivity
- Derived Properties : persistence and recurrence
- Integration in the Polychrony environment for SIGNAL

2nd part – **The SIGNAL Language**

Environment for real-time applications

- Synchronous language, data flow oriented

Signals:

- Infinite sequences of typed values
- Instants of presence: clock

Kernel of the language:

- A:=f(B,C)
- C:=A default B
- (|P1 | P2 |)
- B := A$1 init Bo
- C:=A when B

System of equations

Verification and Synthesis of Controllers: SIGALI

2nd part – Boolean Abstraction – **General Principle 1**

Signals: encoded by three values {-1,0,1}: F_3

$$booléens \Rightarrow \begin{cases} présent \wedge vrai \Rightarrow 1 \\ présent \wedge faux \Rightarrow -1 \\ absent \Rightarrow 0 \end{cases} \qquad non\ booléens \Rightarrow \begin{cases} présent \Rightarrow 1 \\ absent \Rightarrow 0 \end{cases}$$

Operators:

- Synchronization for non Booleans

A:=f(B,C) $\Rightarrow \quad a^2 = b^2 = c^2$

- Synchronization and values for Booleans

C:=A when B $\Rightarrow \quad c = a(-b-b^2)$

- The delay ($): memorization of the previous value in a state variable *x*

B := A$1 init Bo $\Rightarrow \begin{cases} x' = a + (1-a^2)x \\ b = a^2 x \\ x_0 = b_0 \end{cases}$

2nd part – Boolean Abstraction – **General Principle 2**

Signaux booléens	
B := not A	$b = -a$
C := A and B	$c = ab(ab-a-b-1)$ $a^2 = b^2$
C := A or B	$c = ab(1-a-b-ab)$ $a^2 = b^2$
C := A default B	$c = a+(1-a^2)b$
C := A when B	$c = a(-b-b^2)$
B := A $1 (init b_0)	$x' = a+(1-a^2)x$ $b = a^2x$ $x_0 = b_0$
Signaux non booléens	
B := $f(A_1,\ldots,A_n)$	$b^2 = a_1^2 = \cdots = a_n^2$
C := A default B	$c^2 = a^2+b^2-a^2b^2$
C := A when B	$c^2 = a^2(-b-b^2)$
B := A $1 (init b_0)	$b^2 = a^2$

Composition of elementary processes

$$\Downarrow$$

System of polynomial equations in F_3

$$S = \begin{cases} X' = P(X,Y) \\ Q(X,Y) = 0 \\ Q_0(X) = 0 \end{cases}$$

2nd part – Boolean Abstraction – General Principle **example**

```
process Altern = {? event A,B; !}
  ( | C := not ZC
    | ZC := C$1
    | A ^= when C
    | B ^= when ZC
    |)
where boolean C, ZC init false;
```

- Evolution system

$$x' = c + (1 - c^2)x$$

- System of constraints (synchronization)

$$c = -zc,\ a^2 = -c-c^2,\ b^2 = -zc-zc^2$$

$$zc = xc^2$$

- System of initialisation

$$x = -1$$

2nd part – **Polynomial Dynamical Systems**

$$S = \begin{cases} X' = P(X,Y) \\ Q(X,Y) = 0 \\ Q_0(X) = 0 \end{cases}$$

Evolution equations (functions)

Constraint equations (invariant)

Initialisation equations

$X \in F_3^{\ n}$ **State variables**

$Y \in F_3^{\ m}$ **Event variables**

$$P : (Pi)_{i \in [1..n]} : F_3^{\ n+m} \rightarrow F_3^{\ n}$$

$$Q : (Qi)_{i \in [1..m]} : F_3^{\ n+m} \rightarrow F_3$$

$$Q_0 : (Q_{0_i})_{i \in [1..n]} : F_3^{\ n} \rightarrow F_3$$

2nd part – Pol. Dyn. Sys. – **Study of the static part**

$$\begin{cases} Q(X,Y)=0 \end{cases}$$

Constraint equations (invariant)

SIGNAL Clock Calculus

- Solves synchronization constraints
- Structures the control of the application
- Returns a clock hierarchy (forest)

2nd part – Pol. Dyn. Sys. **– Study of the dynamical part**

SIGALI Formal System

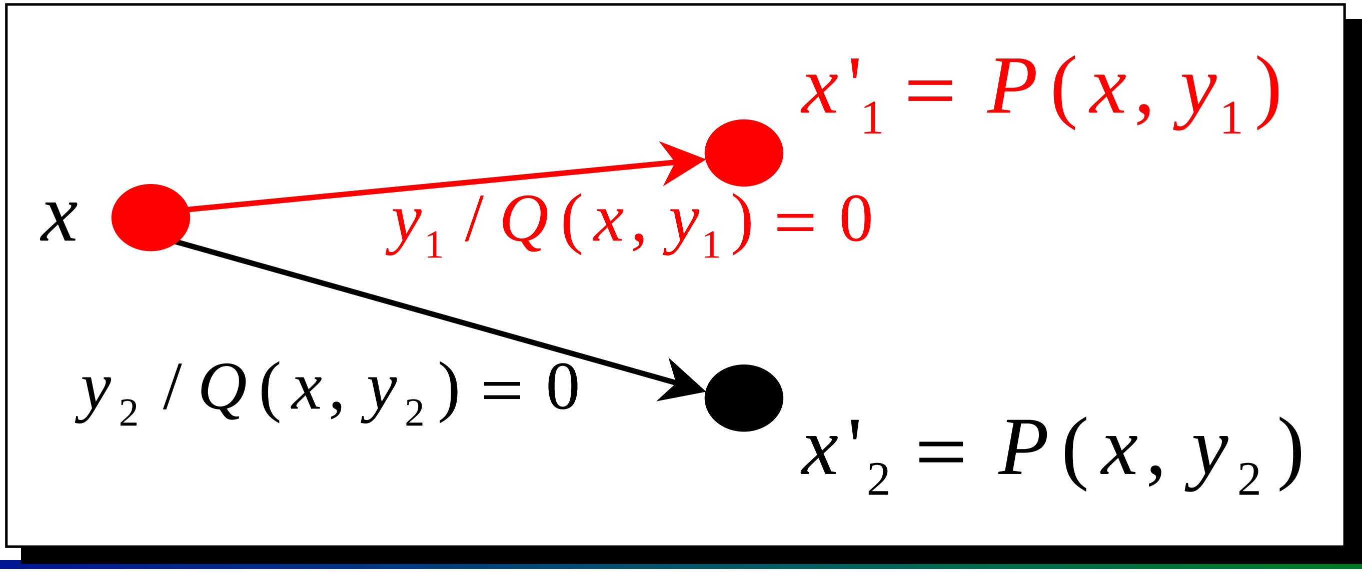
- Equation properties equivalent to the properties expressed on sets of states and events
- Systems of equations represented by ideals of polynomials

2nd part – Pol. Dyn. Sys. – **Underlying explicit automaton**

Initial states:

$$\{x \in F_3^{\,n} \,/\, Q_0(x) = 0\}$$

Evolution:



2nd part – Verification of properties – **Liveness**

Definition

- A state *x* is alive iff there exists an event y admissible in *x*.
- A set of states is alive iff every state is alive.
- A dynamical system is alive iff, for every state *x*, and for every event y s.t. $Q(x,y)=0$, the state $x'=P(x,y)$ is alive.

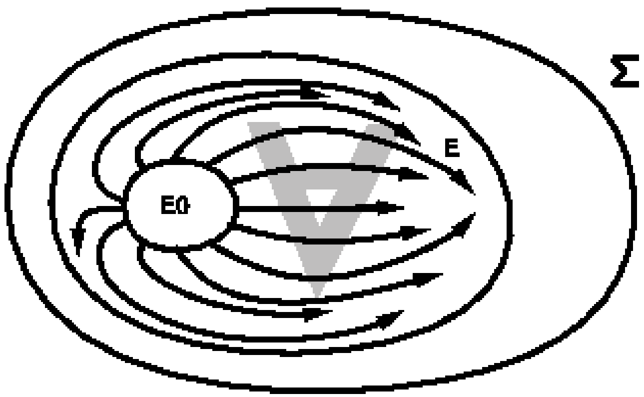
2nd part – Verification of properties – **Invariance**

Definition

A set of states *E* is invariant for a dynamical system, if and only if, for every state *x* of *E*, and for every event y admissible in *x*, the state *x'=P(x,y)* is in *E.*

Set interpretation

$$\forall x \in E, \forall y \in F_3^{\,m}, Q(x,y)=0 \Rightarrow x' = P(x,y) \in E$$



Invariance of *E (from Eo)*

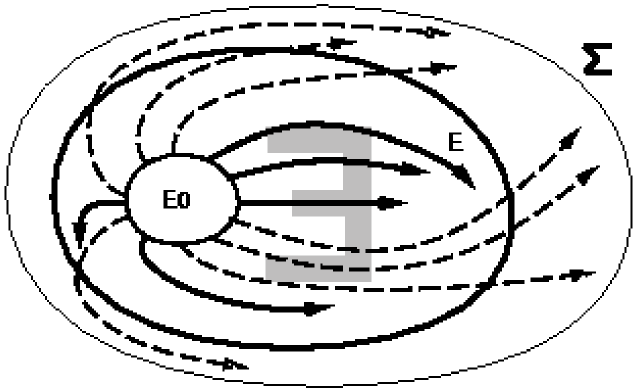
2nd part – Verif. of Prop. – **Invariance under control**

Definition

A set of states *E* is control-invariant for a dynamical system, if and only if, for every state *x* of *E*, there exists an event y admissible in *x*, such that the state *x'=P(x,y)* is in *E*.

Set interpretation

$$\forall x \in E, \exists y \in F_3^{\,m}, Q(x,y) = 0 \text{ et } x' = P(x,y) \in E$$



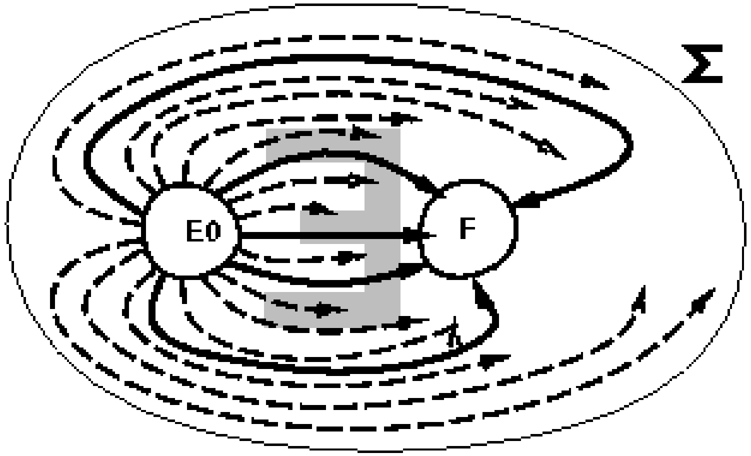
Invariance under control of *E (from Eo)*

2nd part – Verification of properties – **Reach-ability**

Definition

A set of states *F* is reachable for a dynamical system iff every state *x* of *F* can be reached from the initial states *Eo* of the system (i.e., there exists a trajectory initialised in *Eo* that reaches *x*).

Interpretation



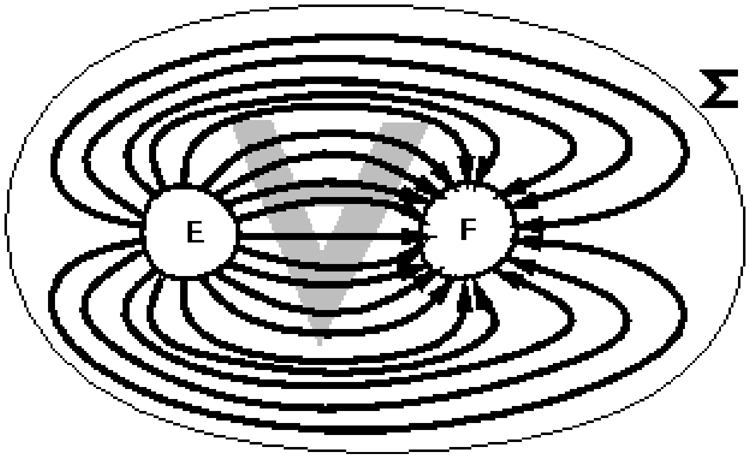
Reach-ability of *F*

2nd part – Verification of properties – **Attractivity**

Definition

A set of states *F* is attractive for a set of states *E* iff every trajectory initialised in *E* reaches *F*.

Interpretation



Attractivity of *F* w.r.t. *E*

2nd part – Verif. of Prop. – **Persistence and recurrence**

Persistence

A set of states is persistent iff it is attractive /reachable from the initial states and if it is also invariant.

Recurrence

A set of states is recurrent iff it is visited infinitely many times.

2nd part – Principle of integration in Polychrony / SIGNAL

Signal Program

System

||

Objectives of Verif. and Control

Polynomial dynamical system

Polychrony

Sigali Program

Sigali

Getting the results of Verif.

Getting the Controllers (TDD)

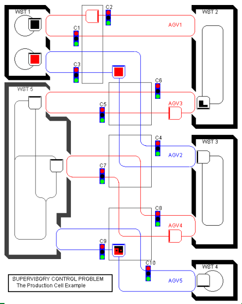
```
(|Sigali(Invariant(Prop))
  |Sigali(S_Reachable(Prop))
  |)
```

System

Solver

Simulation

Signal Compiler



3rd part – **The YATUS**(1) "foot**bridge"**

1) ACCORD/UML Methodology

- Overview of the methodology
- The "Drum Regulator" example

2) Principles of "UML to Signal" Translation

- Signal language target program structure
- Generation rules

(1) **YATUS** : **Y**et **A**nother **T**ranslator from **U**ML to **S**IGNAL

3rd part – UML Methodology - **Overview**

References and tools

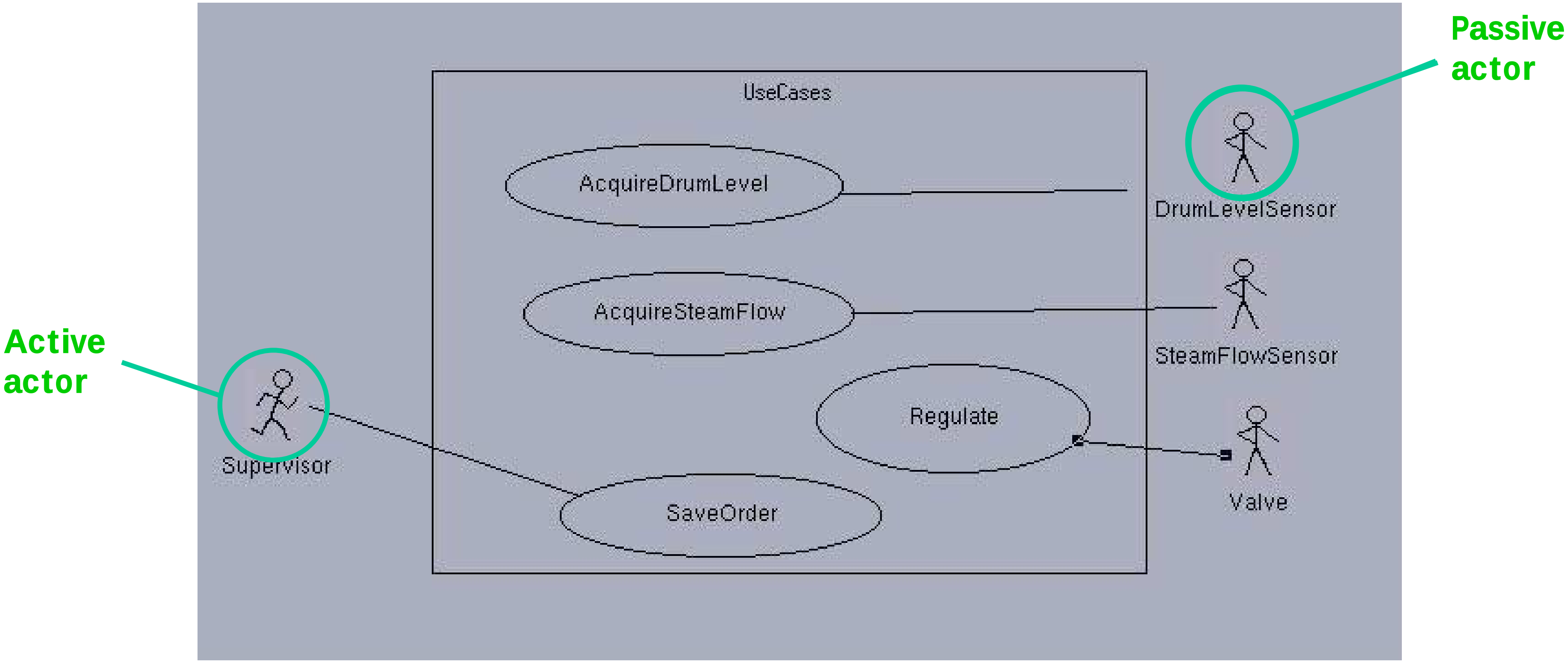
- ACCORD Profile (CEA-LIST)
- Objecteering UML Modeler

Methodology phases

- Preliminary Analysis Model (PAM)
  - Use cases (use case diagrams)
  - High level scenarios (sequence diagrams)
- Detailed Analysis Model (DAM)
  - Structural view (class diagrams)
  - Behavioural view (state-transition diagrams)
  - Interaction view (sequence diagrams)

3rd part – UML Methodology - Overview - **PAM**

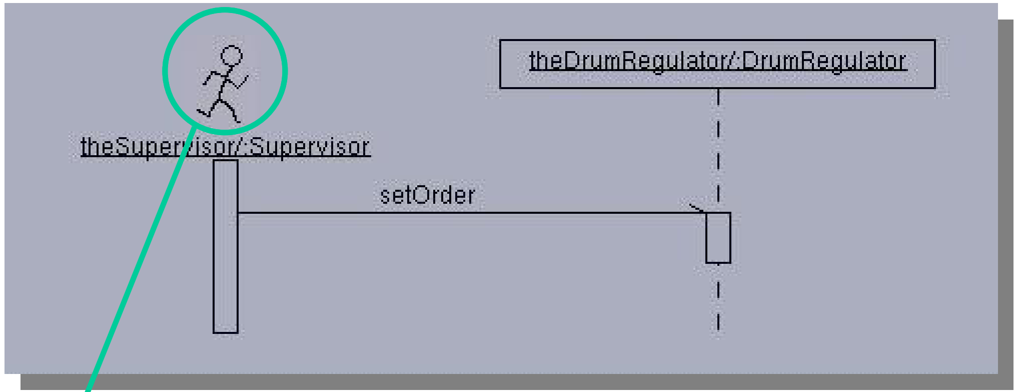
Use case diagram



3rd part – UML Methodology - Overview - **PAM**

High level scenario examples

"SaveOrder" scenario

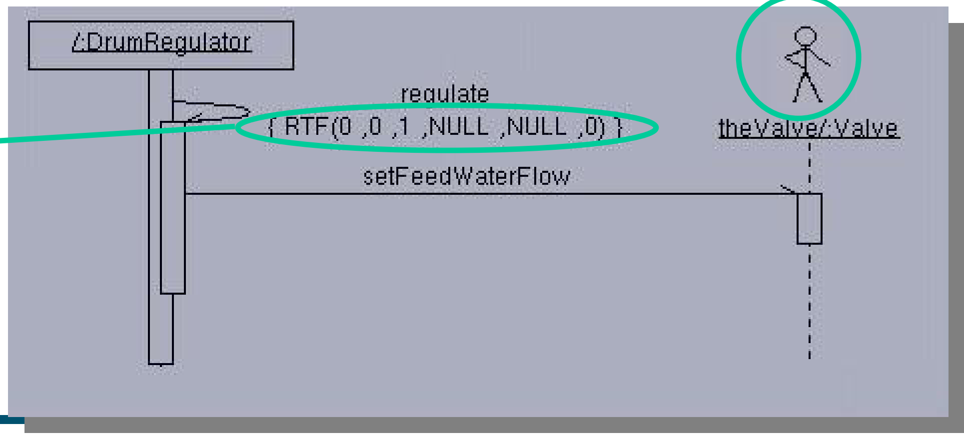


Active actor

"Regulate" scenario

Passive actor

RTF Tagged value for real-time constraints



3rd part – UML Methodology - Overview - **DAM**

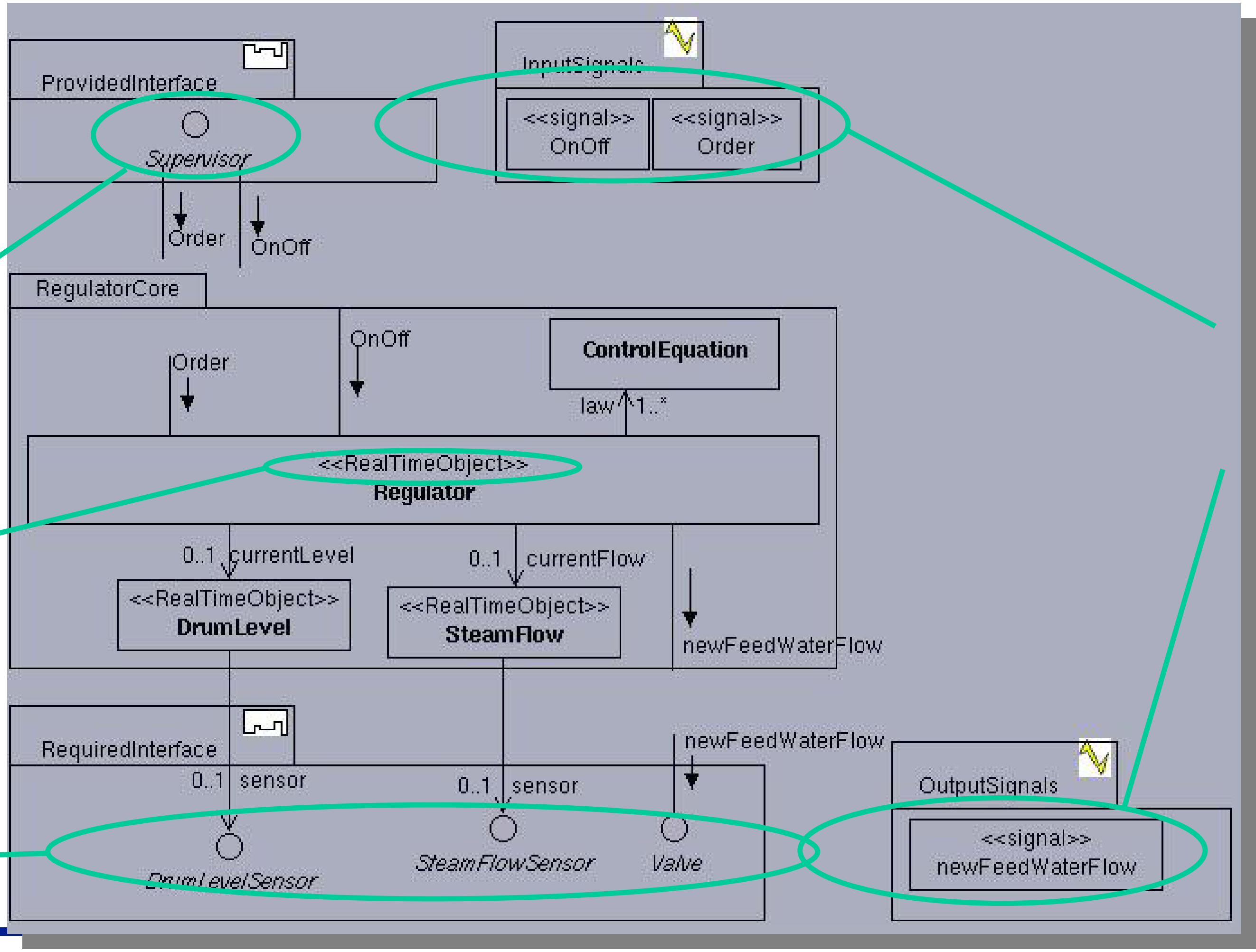
Structural view example

Global class diagram of the system

Transformation of active actors into interfaces

Stereotype for active classes

Transformation of passive actors into interfaces

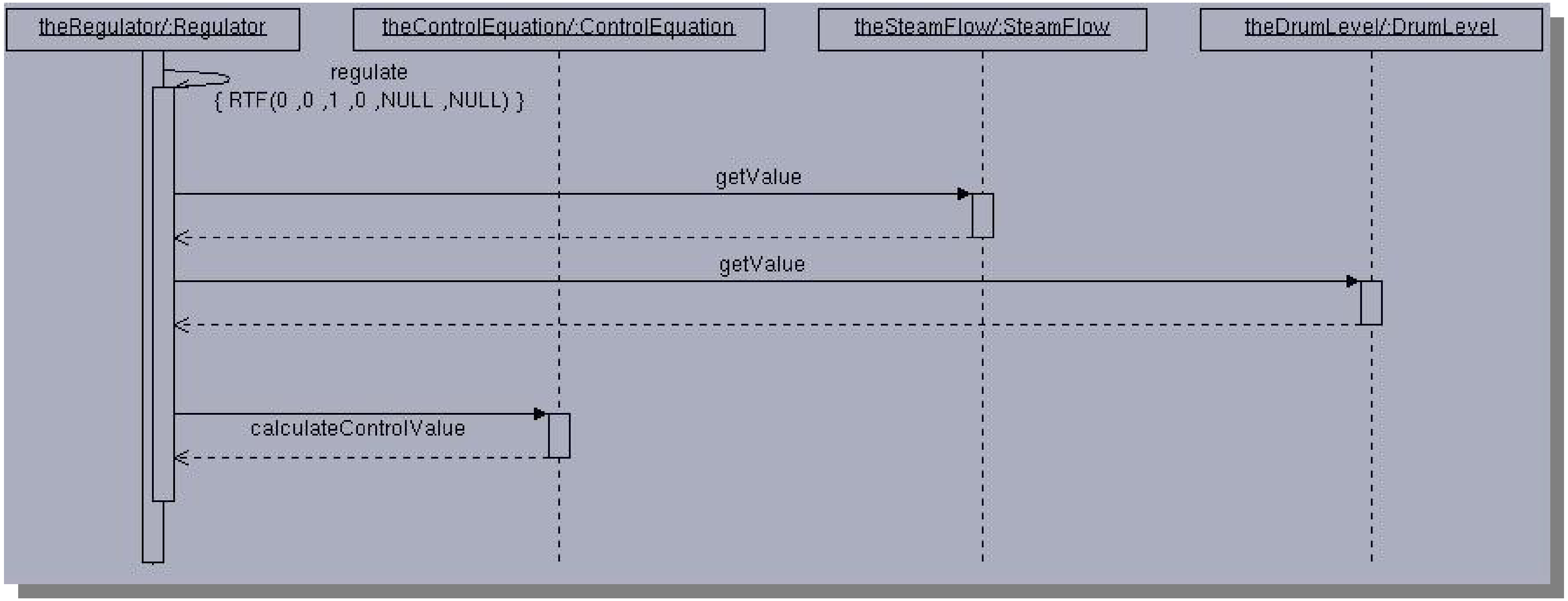


Input/output signals Declaration

3rd part – UML Methodology - Overview - **DAM**

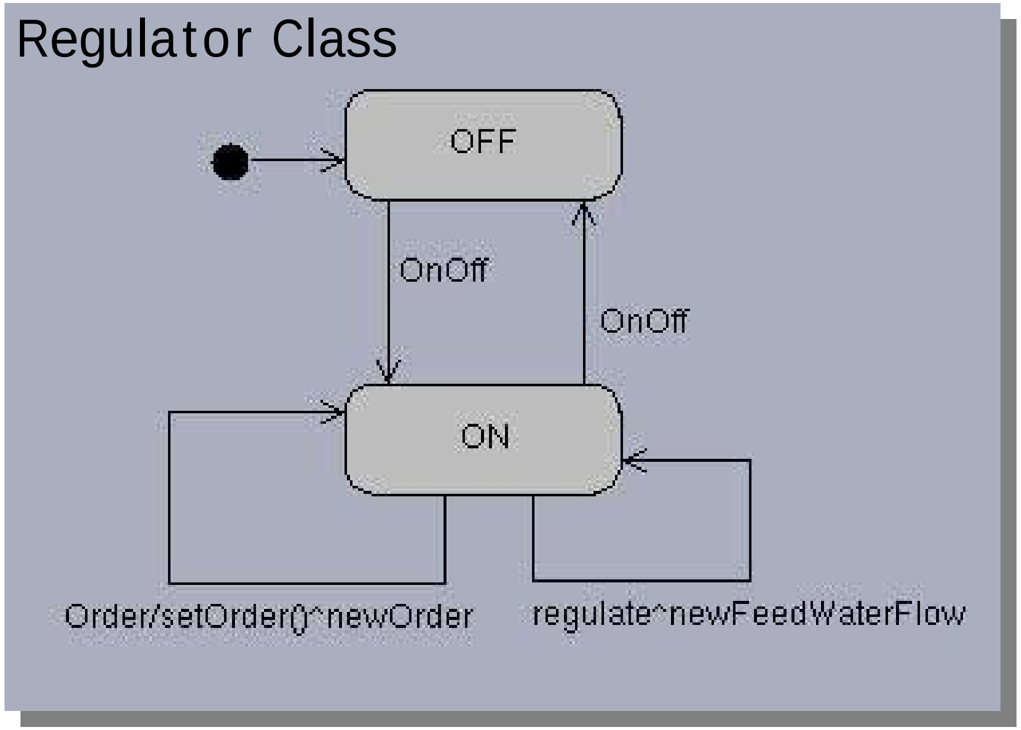
Interaction view example

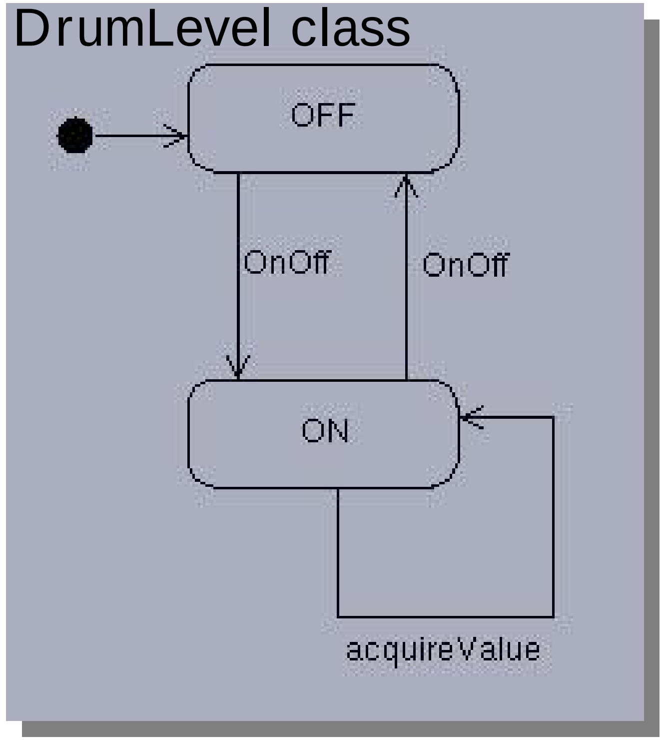
"Regulate" use case detailed scenario



3rd part – UML Methodology - Overview - **DAM**

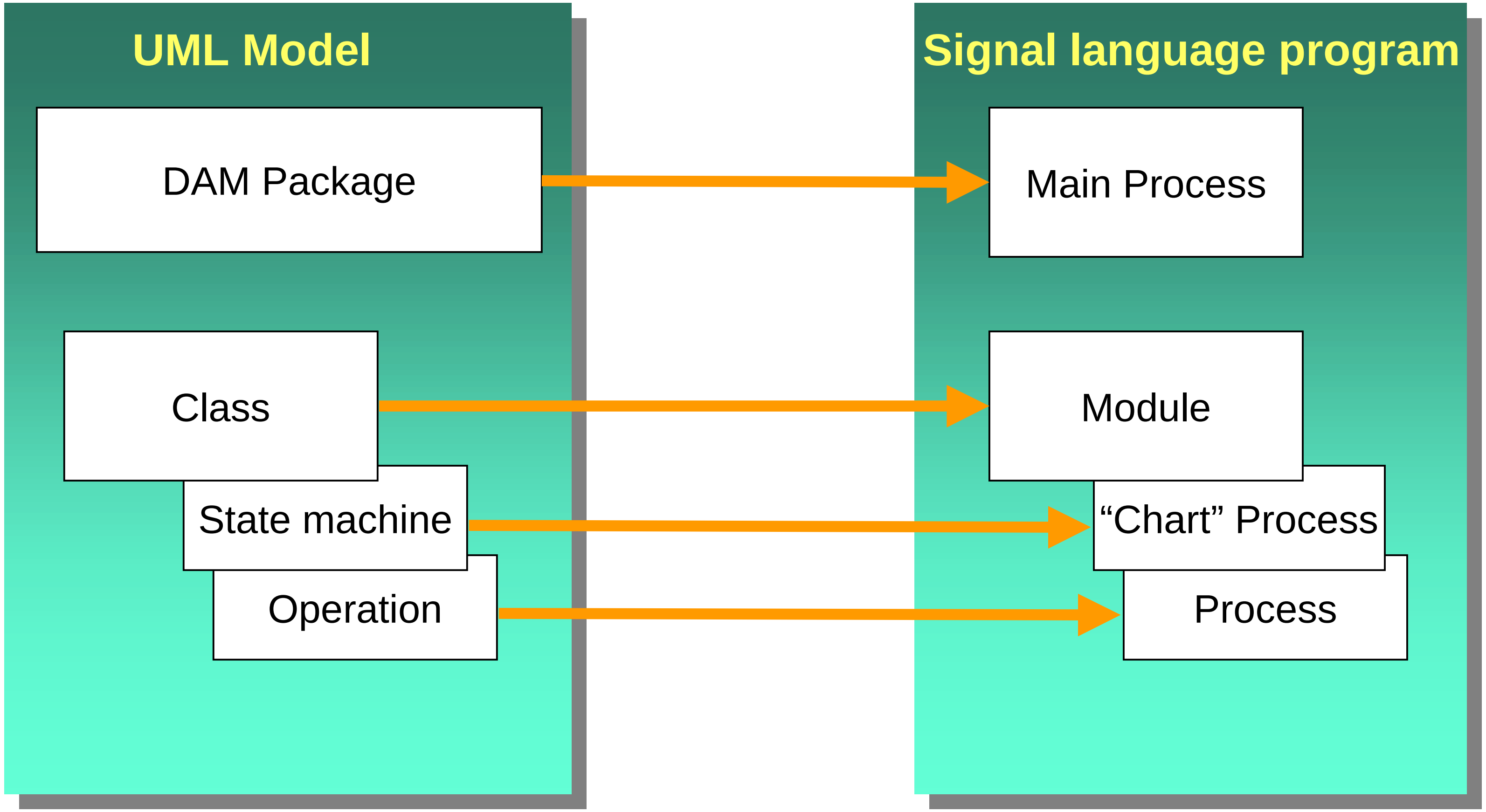
Behavioural view example



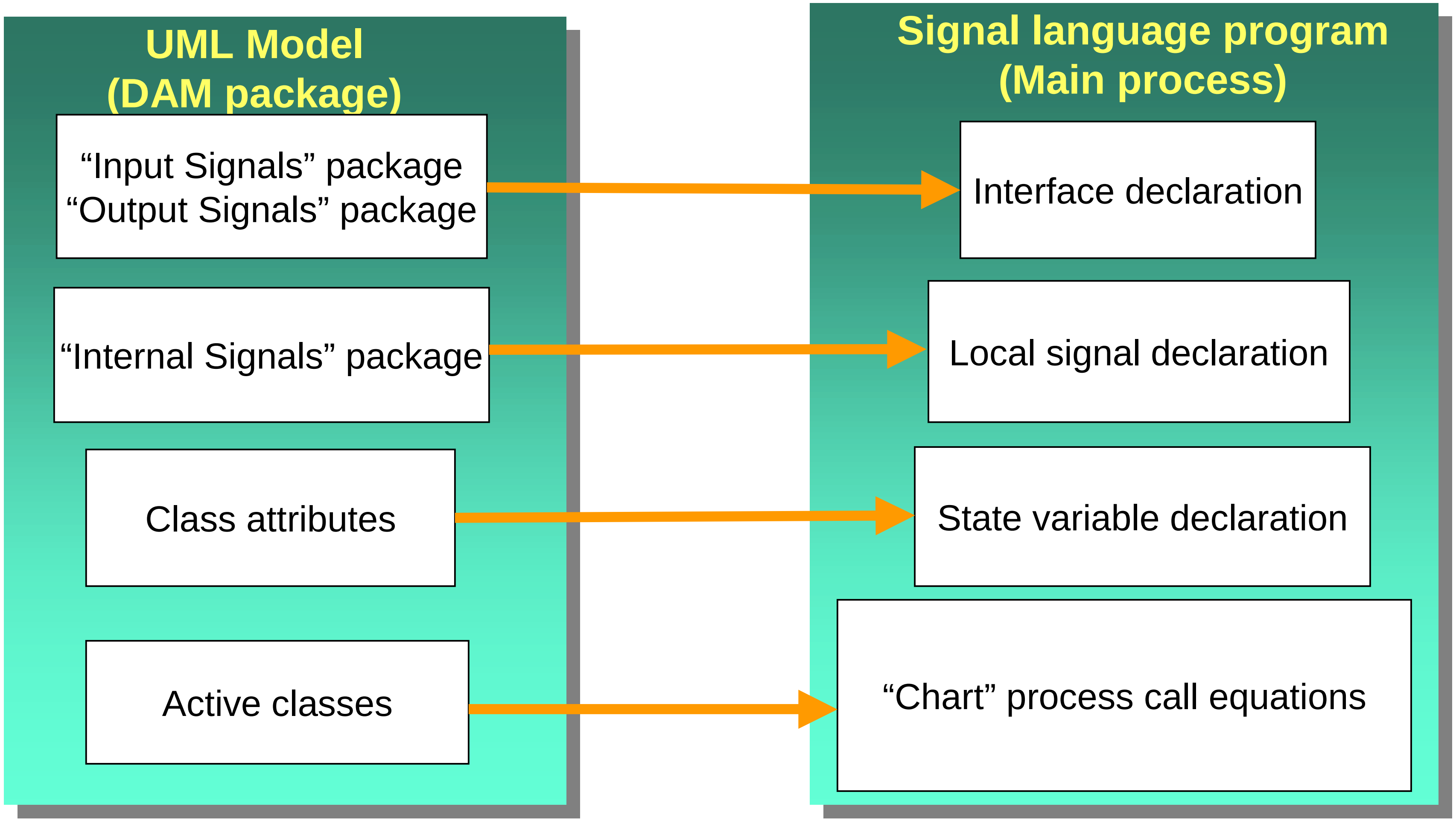


3rd part – **YATUS Principles**

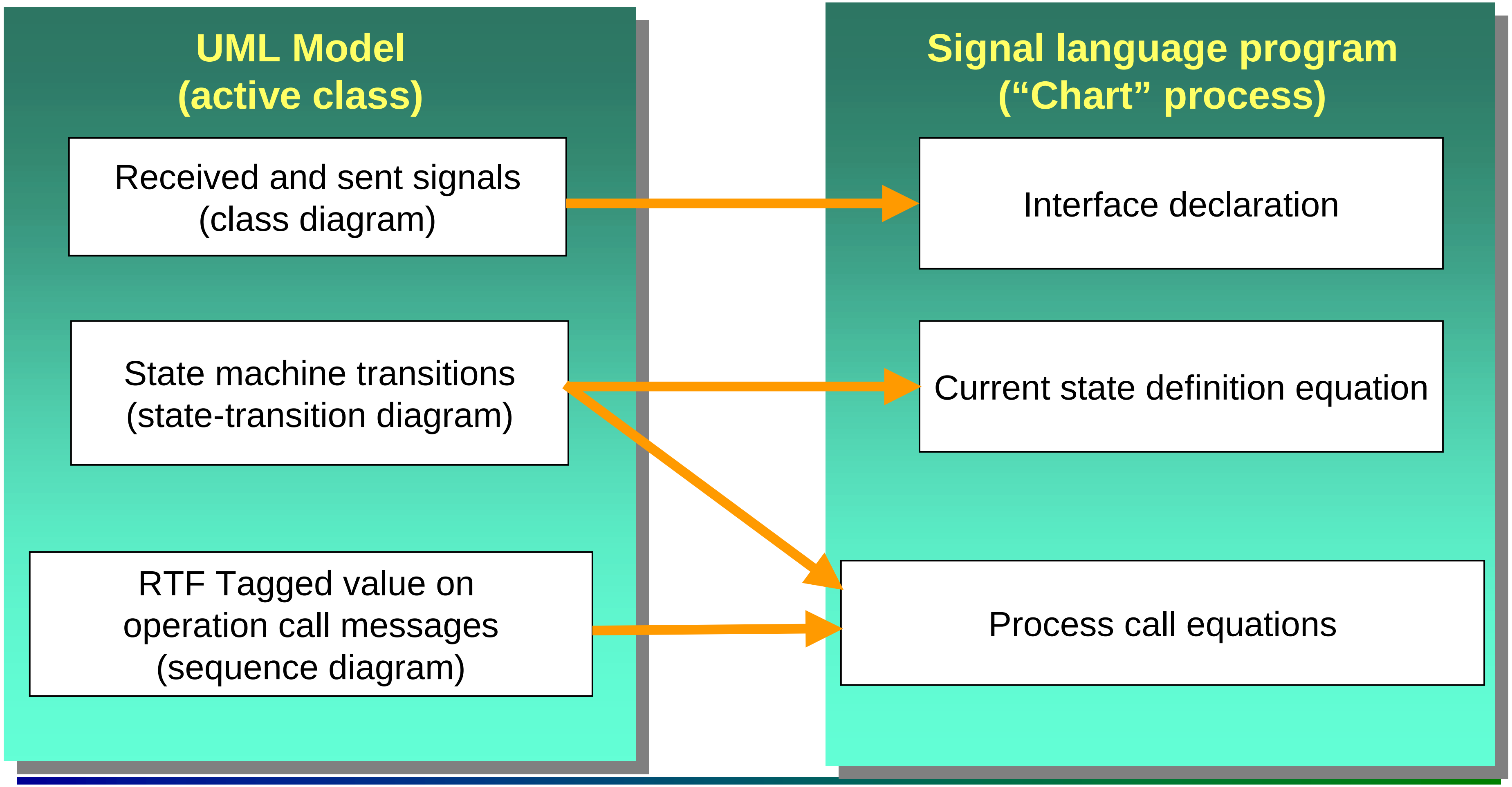
Signal language target program structure



3rd part – **YATUS** : Generation of the "Main" process



3rd part – **YATUS** : Generation of the "Chart" process for an active class



3rd part – **YATUS** : Generation of a process for an operation

UML Model (operation)

Input parameters
Return parameter

→ **Signal language program (process)**

Interface declaration

Note written in Signal language :
"Signal Language Body"
"Signal Language Local Declarations"
"Pragmas Definition"

→

Body + Local declarations

or

Pragmas

3rd part – **YATUS**

Demonstration